Transformation: The Missing Link of MDA

Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel, and Andrew Wood
{agerber,lawley,kerry,steel,woody}@dstc.edu.au

CRC for Enterprise Distributed Systems (DSTC)

Abstract. In this paper we explore the issue of transforming models to models, an essential part of the OMG's Model Driven Architecture (MDA) vision. Drawing from the literature and our experiences implementing a number of transformations using different technologies, we explore the strengths and weaknesses of the different technologies and identify requirements for a transformation language for performing the kind of model-to-model transformations required to realise the MDA vision.

1 Introduction

The OMG's Model Driven Architecture (MDA) [16] defines an approach to enterprise distributed system development that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. The MDA approach envisions mappings from Platform Independent Models (PIMs) to one or more Platform Specific Models (PSMs). The UML Profile for Enterprise Distributed Object Computing (EDOC) [19] represents a first attempt to define a PIM along with several non-normative sketches of mappings to PSMs.

The potential benefits of such an approach are obvious: support for system evolution, high-level models that truly represent and document the implemented system, support for integration and interoperability, and the ability to migrate to new platforms and technologies as they become available.

While technologies such as the Meta Object Facility (MOF) [15] and the UML [17] are well-established foundations on which to build PIMs and PSMs, there is as yet no well-established foundation on which to rely in describing how we take an instance of a PIM and transform it to produce an instance of a PSM.

Our focus is on model-to-model transformations and not with model-to-text transformations. The latter come in to play when taking a final PSM model and using it to produce, for example, Java code or SQL statements. We believe that there are sufficient particular requirements and properties of a model to text transformation, such as templating and boilerplating, that a specialised technology be used. One such technology is Anti-Yacc [7].

Additionally, we will assume that the source and target models are different and leave the problem of update mappings for future work.

In this paper we put MDA to the test, exploring the question of whether we can really describe and perform the required mappings and, if so, then how and with what tools? In Section 2 we begin by surveying some existing related work on model transformation, then in Section 3 we describe a number of our own experiments to express

mappings from the EDOC Business Process model, our source PIM, to the Breeze Workflow model, our target PSM. In Section 4 we identify a number of key requirements of a transformation language based on these experiences and present a model that captures these requirements. Finally, in Section 5, we conclude with a discussion of what we believe are the next steps to take in determining an appropriate foundation for realising the MDA vision.

2 Existing approaches

In this section we examine a number of existing approaches to implementing transformations and discuss their suitability to our goals.

2.1 CWM Transformation

Chapter 13 of the OMG's Common Warehouse Metamodel Specification [2] defines a model for describing Transformations. It supports the concepts of both black-box and white-box transformations. Black-box transformations are not of much interest to us because they only associate source and target elements without describing how one is obtained from the other. White-box transformations, however, describe fine-grained links between source and target elements via the *Transformation* element's association to a *ProcedureExpression*. Unfortunately, because it is a generic model and re-uses concepts from UML, a *ProcedureExpression* can be expressed in any language capable of taking the source element and producing the target element. Thus CWM offers no actual mechanism for implementing transformations, merely a model for describing the existence of a mapping.

2.2 Graph Transformation

Varró et al [24, 6] describe a system for model transformation based on Graph Transformations [1]. In their approach, a transformation consists of a set of rules combined using a number of operators such as sequence, transitive closure, and repeated application. Each rule identifies before and after sub-graphs, where each sub-graph may refer to source and target model elements and associations between them (introduced by the transformation).

This style of approach to model transformation introduces non-determinism in the rule selection, and in the sub-graph selection when applying a rule.

Also, since rules are applied in a sequence, thus resulting in a series of state changes, one needs to be very careful about repeated rule application to ensure termination, and the order of rule application.

2.3 Generated XSLT

Peltier et al. [21, 20] propose that transformation rules are best expressed at the model level and that they should then be translated into a set of rules that operate on the concrete representations of model instances. As such, they propose MOF as the common

meta-model for representing models, XMI [14] as the concrete expression of model instances, and XSLT as the transformation tool to operate on these concrete instances.

Their model for the transformation rules is show as a grammar in Figure 1. Their rules have a mix of both procedural and declarative styles which is in part due to the fact that a given rule may only define a single target element per source element and that target element construction is explicit.

```
rule ::= 'rule' [name] [Entity] 'from' Entity [Restriction]? '{' body '}'
body ::= 'parameters' ':-' ...
'init' ':-' ...
'inherits' ':-' ...
'attributes' ':-' ...
'roles' ':-' ...
```

Fig. 1. Model for an MTRANS transformation rule.

2.4 Text-based tools

Text based tools such as awk and perl are suitable only for the simplest kinds of transformations, largely because they deal with concrete syntax rather than abstract syntax. While arguably more readable and maintainable than XSLT transformations, they require the parsing of input text and serialisation of output text, rather than providing the abstraction of a parse-tree as XSLT does.

3 Our Experiments

In order to identify the requirements for a transformation language suitable for MDA, we primarily attempted to express mappings from the EDOC Business Process model (EDOC-BP) to the Breeze Workflow model using a number of different technologies. Additionally, we attempted a number of other mappings, for example from EDOC-BP to XLANG [23], the underlying model of Microsoft's Biztalk Orchestration, and from Breeze to *dot* [10] allowing for visualisation of the results of mapping from EDOC-BP to Breeze. In the following we describe these attempts and what they taught us.

3.1 XSLT – XMI to XML

Our first attempt at implementing a mapping from EDOC-BP to Breeze used XSLT [25] to map from the XMI representation of the MOF model describing EDOC-BPs to the native XML representation of Breeze workflows [3]. Our motivations for choosing XSLT included the following:

1. both source and target formats were XML,

2. XSLT is based on the concept of matching parts of the source document based on its structure and associating this with the construction of the target document, and
3. a number of XSLT engines are readily available, thus allowing us to focus on describing the mappings rather than having to simultaneously implement a mapping evaluation engine.

Figure 2 shows a fragment of the transformation. This is a relatively simple example that, depending on the attribute `isSynchronous` of the matched OutputGroup or ExceptionGroup either one or another Breeze structure is constructed. Note that ExceptionGroup is a subclass of OutputGroup in the EDOC model, but we cannot exploit this in writing the rule – we must match explicitly on all precise-types.

```
<xsl:template match="ECA.BusinessProcessPkg.OutputGroup |
                     ECA.BusinessProcessPkg.ExceptionGroup">
  <xsl:param name="a"/>
  <xsl:variable name="ct" select="concat(@xmi.id,'.condTask')"/>
  <xsl:choose>
    <xsl:when test="self::node()[@isSynchronous = 'true']">
      <xsl:call-template name="condTaskTemplate">
        <xsl:with-param name="ct" select="$ct"/>
        <xsl:with-param name="a" select="$a"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template
          name="asyncCompoundTaskInputGroupOrActivityOutputGroup">
        <xsl:with-param name="a" select="$a"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Fig. 2. Part of the XSLT for mapping EDOC-BP XMI to Breeze XML.

Not surprisingly, the verbosity of XML as a syntax meant that the mappings became unwieldy and unreadable quite quickly. It was quite difficult to separate the source and target parts of the rules as well as the filtering constraints. In order to write the rules, one needed to be aware of the details of the construction of the syntactic representation of the resulting target model, rather than just its abstract structure. There were also a number of pragmatic problems with the use of XSLT – it is almost impossible to debug, and it is horrendously inefficient since everything is passed by-value which results in a great deal of structure copying.

We also attempted an EDOC-BP to XLANG mapping using XSLT, again because both the source and target instance representations were XML, but this proved somewhat futile due to the overly restricted nature of XLANG as implemented in the beta

version of Biztalk Orchestration available at the time. This meant that the only way to accurately capture the semantics of an EDOC-BP instance using XLANG would have involved embedding the required semantics by generating substantial amounts of code outside of the XLANG model.

3.2 GenGen – a MOF to MOF transformation generator

Having learned that describing mappings at a textual level leads to complications due to the continual transitions from concrete to abstract syntax and back again, we decided to implement our own transformation tool that would operate directly on the model instances involved. That is, we would deal simply with abstract syntax to abstract syntax mappings, leaving the rendering of the result to a separate step (and tool).

The GenGen tool uses mapping rules to generate a Java program that applies these rules to model instances stored in MOF repositories. Its structure and operation is shown in Figure 3. Using DSTC's MOF implementation, dMOF [4], we built model repositories for the mapping rules, the EDOC model and the Breeze workflow model based on MOF meta-model descriptions.

The model for describing the mapping rules was based in part on ideas in the Common Warehouse Metadata model [2] which describes correspondences between MOF model instances.

The generated transformation program would apply the rules one after the other to select objects in a source model repository and create and update objects in the target model repository.

Figure 4 shows the model for these transformation rules. It should be noted that this model has the following characteristics:

1. it has a procedural interpretation; a sequence of steps,
2. target model instances are explicitly created,
3. a traceability relation[1] is constructed and maintained as the rules are evaluated,
4. multiple source model instances can be matched,
5. pre-conditions are used to establish/require correlations between tuples of matching source model instances,
6. multiple target model instances can be created by a rule,
7. the expression language for pre-conditions, post-conditions, and actual transformation is either, one of a small set of pre-defined operations or, a string of Java code which is invoked in the context of the matched elements,
8. arbitrary code can be invoked by a rule.

Figure 5 shows an incomplete fragment of one of the transformation rules[2] [18] parser. Note that this rule maps from a single source model element set, OutputGroup

[1] The traceability relation maintains a record of correspondences between source and target model instances. This can then be used for tasks such as debugging, round-tripping, and update propagation.

[2] The OMG's Human-Usable Textual Notation specification allows for the configurable definition of textual languages for MOF models. When prototyping tools such as GenGen, we use TokTok [5] to automatically generate a HUTN and associated parser.

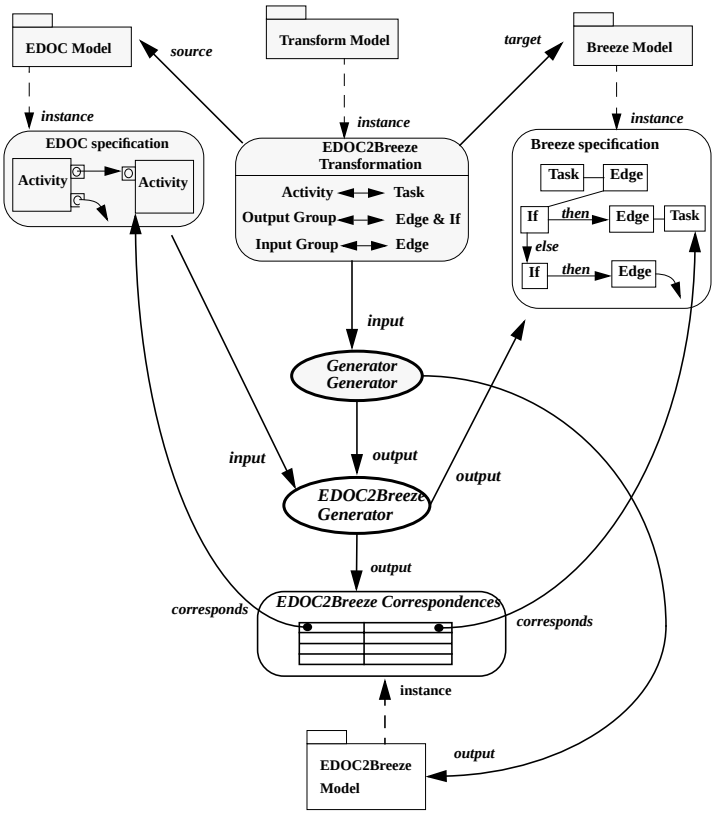

Fig. 3. Applying GenGen to produce an EDOC2Breeze transformer.

instances, to two target model class instances and an association. It also defined values for attributes of these instances. It contrasts with the XSLT rule in Figure 2 in that the set of source model instances that match this rule could include ExceptionGroup instances simply by specifying the `use_subtypes` attribute.

The GenGen experiment was quite successful and led to the discovery of the need for non-uniform or asymmetric transformations. These are transformations where an arbitrarily chosen subset (usually just one) of the elements that match a source pattern is mapped differently to all the others. You can imagine this happening in the generation of something like cascading if-then-else statements. The last match in the transformation either has no *if* condition or no subsequent *else*-clause.

Using Java code fragments for expressing conditions and transformation functions leads to difficulties in expressing the transformations since it requires detailed under-

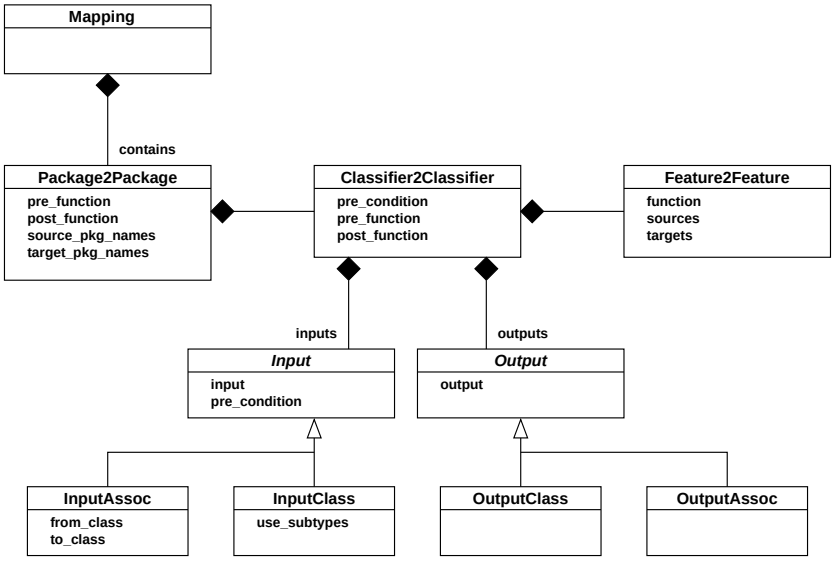

Fig. 4. The GenGen transformation model.

standing of how these fragments will be embedded in the generated code. In effect, the GenGen tool embodies a mapping from a hybrid transformation-rule/Java model to the Java language model and knowledge of this mapping is required to express the mapping rules.

After implementing GenGen, we felt that the transformation rules were too procedural and that the declarative nature and unification capabilities of a logic-programming style of transformation rule might be more suitable for describing and implementing mappings.

3.3 Mercury – declarative transformation using logic programming

We chose to use the Mercury programming language [22] for a number of reasons. Being a purely declarative logic language, it provided us with unification for pattern matching the source instance-graph. It is also a strongly-typed language with an efficient implementation based on compilation to C and it is advertised as supporting a CORBA IDL mapping [8] which we hoped would allow future connection to our MOF repositories.

The style of transformation we adopted involved writing predicates defining individual pieces of the target model instance in terms of the source model instance. In essence, context-free sub-transformation rules were written, then rules that established context (for example, ensuring an association existed between two source model instances) would invoke the more general sub-transformation rules. This style of writing rules lead to repetitive rule bodies and fragmented transformations; it became difficult to group rules based on either the set of things generated from a given source model

```
Classifier2Classifier "FromOutputGroup" {
  InputClass  { input: "OutputGroup" }
  OutputClass { output: "Task" }
  Feature2Feature {
    sources: ("name")
    targets: ("target0_name")
    function: "COPY"
  }
  Feature2Feature {
    targets: ("target0_and_join" "target1_and_join")
    function: "CONSTANT false"
  }
  OutputClass{ output: "ConditionalTask" }
  Feature2Feature {
    targets: ("target1_name")
    function: ‘target1_name=source0_name+"_test";‘
  }
  Feature2Feature {
    sources: ("name")
    targets: ("expression")
    function:
      ‘target1_expression = "flag == " + ’"’ + source0_name + ’"’;‘
  }
  OutputAssoc { output: "conditional_then" }
  Feature2Feature {
    targets: ("cond")
    function:‘target2_cond=target1;‘
  }
}
```

Fig. 5. Incomplete fragment of a GenGen rule for mapping EDOC-BP to Breeze.

element, or the set of source model elements contributing to the generation of a given target model element.

An example of the style of rules is shown in Figure 6. Note that this is a very small fragment of the transformation and that certain unimportant details have been elided.

Since Mercury’s type system offers a form of inheritance, *typeclasses*, we attempted to use it to capture the inheritance semantics of our source and target models. However, typeclasses do not provide traditional constructive or aggregation-based inheritance. Instead, they allow common structure in several types to be identified and then used in polymorphic rules. This did not suit our desire to capture the inheritance semantics of the source and target models.

The major lessons learned from the Mercury implementation were that the models, instances, and the meta-model should all be explicitly represented so that the semantics of the models and their instances are available to be used when writing the rules, and that the ability to define multiple targets in a single rule leads to a much more compact and, presumably, more readable set of rules since it allows for greater modularity.

```
conditionaltask(Id) :-
        conditionaltask_for_outputgroup_of_activity(Id, _OutputGroup).

conditionaltask_for_outputgroup_of_activity(Id, OG) :-
        outputgroup_of_activity(OG, _Activity),
        mapId(OGˆog_id, conditionaltask_for_outputgroup, Id).

outputgroup_of_activity(OutputGroup, Activity) :-
        outputgroup(OutputGroup),
        contains(Activityˆa_id, OutputGroupˆog_id),
        activity(Activity).
```

Fig. 6. Part of the Mercury rules for mapping EDOC-BP to Breeze.

3.4 F-Logic – declarative transformation based on object-oriented logic programming

F-Logic [9] is one of the most developed and complete formal models for deductive object-oriented languages. Its features include object identity, complex objects, inheritance, polymorphic types, query methods, and encapsulation.

Our primary motivation for turning to F-Logic stemmed from our frustrations with trying to use Mercury's type system to reflect the semantics of the MOF meta-model. However, it also offered a number of other advantages: a very flexible and compact syntax for defining rules that could be interpreted at both the model and instance levels, and the ability to define multiple targets in a single rule. For example, the set of facts:

```
o1 : workflow.
o2 : task.
o3 : task.
o1[name -> "Quokka Example"].
o1[tasks ->> o2].
o1[tasks ->> o3].
o2[name -> "Find quokka"].
o3[name -> "Feed quokka"].
```

can also be written quite compactly as:

```
o1 : workflow [
  name -> "Quokka Example",
  tasks ->> {
    o2 : task [name -> "Find quokka"],
    o3 : task [name -> "Feed quokka"]
  }
].
```

and this compact *molecule* representation can be used in both rule heads and rule bodies allowing rules such as:

```
wf(CT) : workflow [
  tasks ->> t(A) : task[join -> orJoin]
] :-
    CT:compoundTask[
      contains ->> A:activity
    ].
```

which defines two object instances (`wf(CT)` and `t(A)` of classes `workflow` and `task` respectively), an association between the workflow instance and the task instance, and the value (`orJoin`) of an attribute of the task. Without the option of molecular or multiple rule heads, this would have required 4 separate rules with mostly-repeated rule-bodies.

Using this approach we were able to describe an EDOC-BP to Breeze mapping using only 8 transformation rules in addition to the rules that described the MOF-based models. A by-product of this approach is that, where one would normally have grouped rules for readability, the F-Logic syntax allows and encourages rules to be combined. The resulting rules then tend to a form where a set of source model elements are matched based on type, association, and other constraints (attribute values, for example), and a set of resulting target model elements are defined in terms of them.

Interestingly, this grouping of multiple context-related rules into larger rules tends to mirror the kinds of diagrams we would sketch on white-boards to communicate these mappings to one-another. That is, in describing a mapping to a colleague, we would rarely say this element maps to that element, this association to that association and element, etc. Instead, we would say "when you have this arrangement (pattern) of source elements, you get this other arrangement of target elements where this thing corresponds to that thing, etc." (see Figure 7).

The elements of the F-Logic implementation that proved beneficial were:

1. explicit naming of variables,
2. the ability to define multiple targets in a single rule, and
3. the ability to define these targets with differing cardinalities.

For our first attempt at using F-Logic we used the Flora [11] Prolog generating compiler which is an alpha quality implementation. This had two major consequences: 1) it was terribly inefficient, requiring 1.5G of RAM to perform the EDOC-BP-to-Breeze-to-*dot* mapping, and 2) its support for negation was problematic.

4 MDA Mapping Language Requirements

Having experimented with our own transformations and examined the experiments of others, we now have a set of requirements for a transformation language suitable for describing in a precise but readable manner, the kinds of model to model mapping rules required to realise the MDA vision.

4.1 Functional Requirements

The transformation language must be able to:

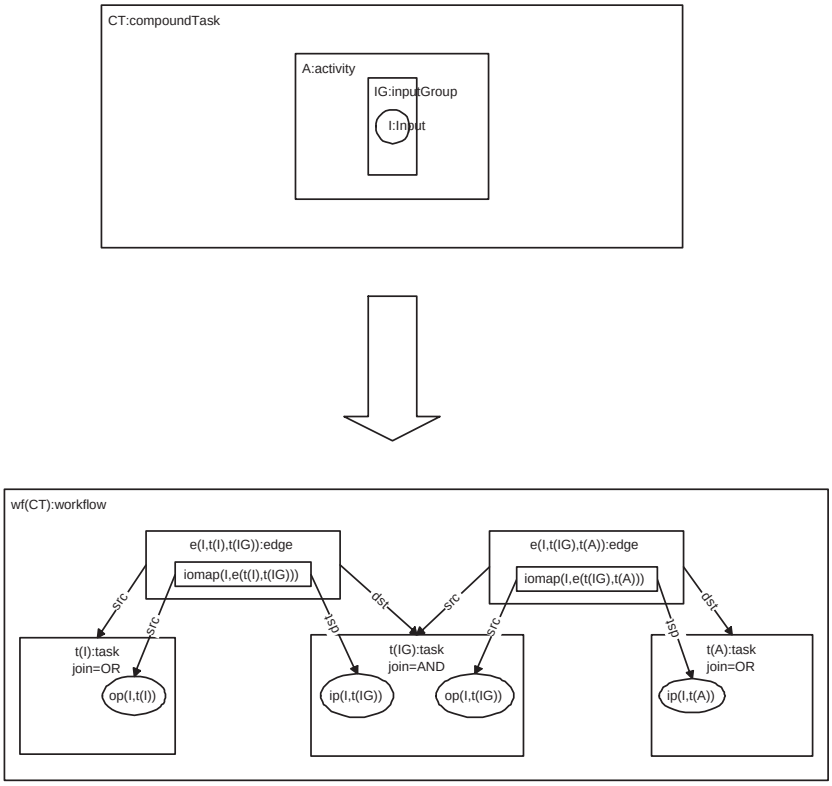

Fig. 7. Pictorial description of a mapping rule, as might be used for communication between people.

1. Match sets of source model elements.
2. Match elements by type (include instances of sub-types) and precise-type (exclude instances of sub-types). The mapping for an EDOC ExceptionGroup is different to the mapping for its concrete supertype, OutputGroup, for example.
3. Filter the set of matched tuples based on associations, attribute values, and other criteria. For example, an EDOC Input contained by an InputGroup is mapped differently to an Input contained by an Activity.
4. Establish associations between source and target model elements, possibly implicitly. These associations can then be used for maintaining traceability information.
5. Define different mappings for the first and last elements of the set of matched tuples. For example, when generating cascading if-then-else structures. More generally, matched elements may require a stable total ordering, which could be purely arbitrary, so that mapping rules can identify the successor, predecessor, and index of an element and whether it is the first or last in the set. These abilities are required when, for example, populating a table-like model with matched elements.
6. Handle recursive structure with arbitrary levels of nesting. For example, the uniqueness semantics of the source and target models may differ requiring the construction

of *fully-qualified* names with a global scope in the target model from locally-scoped names in the source model.

4.2 Usability Requirements

It is desirable for readability and expressiveness concerns that:

1. multiple target elements are definable in a single rule,
2. rules are able to be grouped naturally for readability and modularity,
3. intermediate transformations should be definable, thus supporting multi-step transformations,
4. embedding of conditions and expressions in the transformation model is explicit and seamless, and
5. optional attributes are easily dealt with.

Reflecting on the transformation model used by GenGen and the implicit model used in the F-Logic based transformation, one can see they are quite similar. The major points of departure being the use of explicitly named variable bindings, varied multiplicities for generated elements, and the embedding of the condition/expression language.

Based on these requirements and a declarative execution model that involves matching a set of tuples or source model elements, filtering the set, and defining a set of resulting target model elements and their associated attributes, we have developed the transformation model shown in Figure 8.

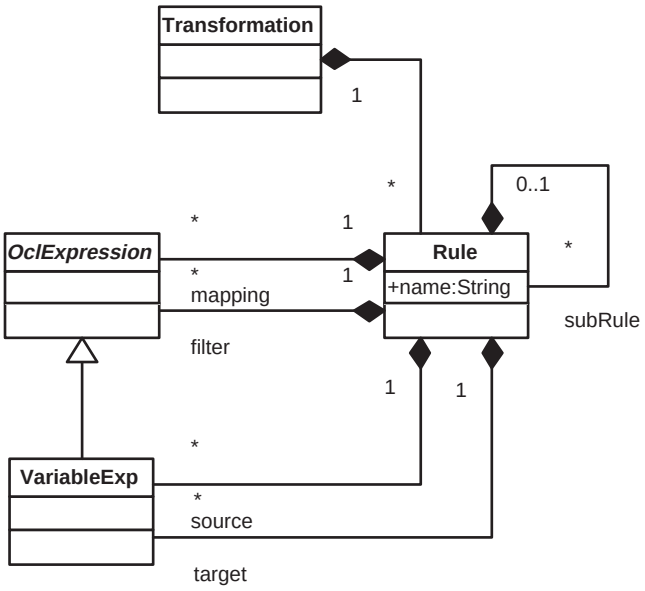

Fig. 8. Core aspects of proposed mapping model.

With this model we are anticipating using the model for the Object Constraint Language (OCL) 2.0 [13] as the basis for our expression language.

The model explicitly identifies source elements, target elements, a filter expression, and a mapping expression.

Figure 9 shows an example source and target model while Figure 10 gives an example of a mapping rule using possible concrete syntax for the model. Note the use of the nested rule to produce a set of Baz elements per Bar element.

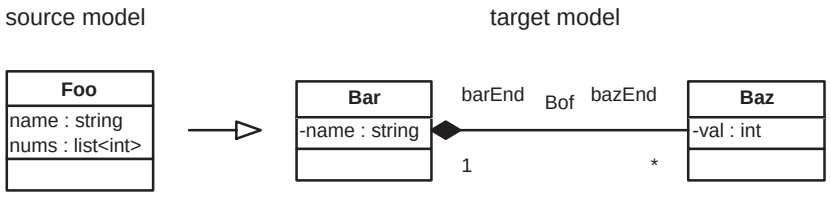

Fig. 9. Example source and target models.

```
r1 (x : Foo) → (y : Bar) ::=
        y.name = x.name
        r2 (a : x.nums) → (z : Baz, a : Bof)
                z.val = a, a.barEnd = y, a.bazEnd = z
```

Fig. 10. Example mapping including a nested rule.

Implementation of a transformation engine supporting the model of Figure 8 remains to be done. We anticipate beginning with an F-Logic or other logic-programming based prototype to sort out the details before adapting the GenGen tool to this model.

5 Conclusion

In carrying out our own transformations and examining the efforts of others it becomes clear that there are two quite different styles of transformation: procedural, with explicit source model traversal and target object creation and update, and declarative, with implicit source model traversal and implicit target object creation/virtual target objects.

Currently, we tend to a preference for the declarative approach due to the simpler semantic model required to understand the transformation rules – order of rule application and termination semantics are a non-issue with the declarative approach. However, it should also be noted that when considering transformations that update a model, something that is outside the scope of this paper, some form of procedural aspect may be necessary.

In section 4 we identified a number of functional and non-functional requirements for an MDA transformation language. Amongst these, the need to be able to define non-uniform mappings and to define multiple targets with different cardinalities are notable.

We have indicated the desirability to be able to define and perform multi-step transformations, or to define intermediate rules that may define elements that are not part of the target model.

We have not discussed the issue of optimisation of transformations, although our XSLT mapping from EDOC-BP XMI to Breeze XML did include a second set of transformation rules that would map from the output of the first set of simple rules and remove any redundant elements produced by that mapping. Indeed, there is nothing special about an optimisation step other than the source and target models being the same, and the consequent requirement that the transformation rules be able to distinguish between source and target instances.

In defining mappings from model to model, the question of correctness of the mapping arises. There are several notions of correctness that can be considered. The simplest is that of syntactic or structural correctness. That is, given a well-formed instance of the source model, is it always the case that a well-formed instance of the target model is a result of the mapping? The more complex form of correctness is that of semantic correctness; does the result of the transformation *mean* the same thing as the input? One must avoid the implicit assumption, in both of these cases, that only complete or consistent models are of interest. There are cases, such as when transforming a model instance to some visual representation, where this may not be the case.

While our work has not investigated such theoretical issues, we did find it valuable to be able to express consistency criteria specific either to our source and target models, or to the particular transformation that we were implementing when working with both Mercury and the F-Logic implementations. Additionally, limited structural consistency checking can be performed using DTDs for XML-based transformations such as XMI and Breeze XML.

With regards the generic issue of semantic correctness, it is complicated by the fact that the source and target instances are, usually, statements in different domains of discourse and that the complexity of defining correspondences between statements in these two domains is itself tantamount to defining the transformation we wish to check. Thus this is only ever likely to be a feasible task when the source and target models are the same, isomorphic, or only very slightly different.

In conclusion, the MOF 2.0 Query/Views/Transformations RFP, not yet issued, will be the 6th in a series of MOF 2.0 RFPs [12] and has the potential to play a key role in the MDA vision. It is imperative for the success of this vision that an effective means of defining model transformations is developed. Unless such a solution addresses requirements such as those identified in this paper, transformation will remain the missing link of MDA.

6 Acknowledgments

The work reported in this paper has been funded in part by the Co-operative Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Education, Science and Training).

References

[1] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Pump, A. Schürr, and G. Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 34(1):1–54, Apr. 1999.

[2] CWM Partners. Common Warehouse Metamodel (CWM) Specification. OMG Documents: ad/01-02-{01,02,03}, Feb. 2001.

[3] DSTC. Breeze: Workflow with ease, online documentation. `http://www.dstc.edu.au/Research/Projects/Pegamento/Breeze/breeze.html`.

[4] DSTC. dMOF: an OMG Meta-Object Facility Implementation. `http://www.dstc.edu.au/Products/CORBA/MOF/index.html`.

[5] DSTC. TokTok – The Language Generator. `http://www.dstc.edu.au/Research/Projects/Pegamento/TokTo`

[6] S. Gyapay and D. Varró. Automatic Algorithm Generation for Visual Control Structures. Technical report, Dept. of Measure,ment and Information Systems, Budapest University of Technology and Economics, Dec. 2000. `http://www.inf.mit.bme.hu/FTSRG/Publications/TR-12-2000.pdf`.

[7] D. Hearnden and K. Raymond. Anti-Yacc: MOF-to-text. Submitted to EDOC 2002.

[8] D. Jeffery, T. Dowd, and Z. Somogyi. MCORBA: A CORBA Binding for Mercury. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, volume 1551 of *Lecture Notes in Computer Science*, pages 211–227, San Antonio, Texas, Jan. 1999. Springer Verlag.

[9] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, July 1995.

[10] E. Koutsofios and S. North. Drawing graphs with *dot*. `http://www.research.att.com/sw/tools/graphviz/dotguide.pdf`, Feb. 2002.

[11] B. Lundäscher, G. Yang, and M. Kifer. FLORA: The Secret of Object-Oriented Logic Programming. Technoical report, SUNY at Stony Brook, 1999.

[12] Request for Proposal: MOF 2.0 Core RFP. OMG Document: ad/01-11-05, Nov. 2001.

[13] Request for Proposal: UML 2.0 OCL RFP. OMG Document: ad/00-09-03, Sept. 2000.

[14] OMG. Interchange Metamodel in XML. OMG Document: formal/01-02-15, Feb. 2001.

[15] OMG. Meta Object Facility (MOF) v1.3.1. OMG Document: formal/01-11-02, Nov. 2001.

[16] OMG. Model Driven Architecture – A Technical Perspective. OMG Document: ormsc/01-07-01, July 2001.

[17] OMG. Unified Modeling Language v1.4. OMG Document: formal/01-09-67, Sept. 2001.

[18] OMG. Human-Usable Textual Notation. OMG Document: ad/02-03-02, Apr. 2002.

[19] OMG. UML Profile for Enterprise Distributed Object Computing (EDOC). OMG Document: ptc/02-02-05, Feb. 2002.

[20] M. Peltier, J. Bézivin, and G. Guillaume. MTRANS: A general framework, based on XSLT, for model transformations. In *WTUML'01, Proceedsings of the Workshop on Transformations in UML*, Genova, Italy, Apr. 2001.

[21] M. Peltier, F. Ziserman, and J. Bézivin. On levels of model transformation. In *XML Europe 2000*, pages 1–17, Paris, France, June 2000. Graphic Communications Association.

[22] Z. Somogyi, F. Henderson, and T. Conway. Mercury: an efficient purely declarative logic programming language. In *Proceedings of the Australian Computer Science Conference*, pages 499–512, Glenelg, Australia, Feb. 1995.

[23] S. Thatte. XLANG Web Services for Business Process Design. Microsoft: `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`, 2001.

[24] D. Varró, G. Varraó, and A. Pataricza. Designing the Automatic Transformation of Visual Languages. Accepted for Science of Computer Programming.

[25] W3C. XSL Transformations (XSLT) v1.0. W3C Recommendation: `http://www.w3.org/TR/xslt`, Nov. 1999.